1)

TYPE

medico = record

nombreyApellido:string[100];

dni:integer;

matricula:real;

anoIngreso:integer;

end;

nodo = record

claves:array[M-1] of medico;

hijos: array[M] of integer;

cantElementos:integer;

end;

file of nodo = arbol;

Como clave de identificación hay que usar el DNI, ya que es la única clave univoca que tiene el registro médico. Se podría seleccionar las otras como claves para índices secundarios, pero como se pueden repetir las claves, es necesario tener una clave unívoca que de otro orden en caso de que se repitan.

2)

TYPE

medico = record

nombreyApellido: string [100];

dni: integer;

matricula: real;

anoIngreso: integer;

end;

nodo = record

claves: array[M-1] of medico;

hijos: array[M] of integer;

cantElementos: integer;

nodo\_sig: ^nodo;

end;

file of nodo = arbol;

1. Voy comparando desde la raíz hasta llegar a los nodos terminales (donde se encuentran los datos), a través del índice principal ya que este se encuentra ordenado por DNI, y así obtengo la dirección efectiva del registro buscado del archivo de datos.
2. Utilizando un índice ordenado por apellido y nombre, de este se obtiene la clave primaria para acceder al índice primario, y de este la dirección efectiva del registro buscado (García Mariano). O si no se puede tener el índice secundario también con la referencia al elemento del archivo como el índice principal, pero si se fuera a modificar la dirección de un elemento se tendrían que modificar todos los índices y no solo el principal.
3. Al tener los elementos como nodos terminales en los árboles B+ vinculados entre ellos, se facilita la búsqueda secuencial de varios elementos, permitiendo obtener los elementos dado por el orden que tenga el arbol.

4)

1. **PosicionaryLeerNodo ()** recibe el árbol A (archivo), hace un seek (A, NRR) y luego un read (A, nodo) para leer el nodo y tenerlo en una variable. El árbol A va por referencia ya que es un archivo, NRR va por valor ya que no se modifica y nodo va por referencia ya que se tiene que leer el nuevo nodo por lo que se debe modificar.
2. **claveEncontrada ()** Recorre el array de elementos buscando la clave. Si el elemento leído es menor, pasa al siguiente elemento, si no hay más queda en pos el hijo derecho del último elemento (el mayor de todos). Si el elemento leído es mayor, queda en pos el hijo izquierdo de ese elemento. En los 2 casos anteriores se retorna false, si se encuentra la clave se retorna true.

A (archivo-árbol) se debe pasa por referencia como es un archivo, nodo pasa por valor ya que no se modifica al igual que la clave, y pos por referencia ya que si es necesario modificarlo en caso de que no se encuentre la clave y se quiera referenciar a otro nodo.

1. No se pasa el árbol por la función buscar, y la variable nodo no está declarada en la función (no existe). Debería quedar:

function buscar (A, NRR, clave, NRR\_encontrado, pos\_encontrada)

Var

Nodo: nodo;

5)

* Un **Overflow** en arboles se da cuando se tratan de insertar un elemento en un nodo que ya está lleno por lo que no tiene capacidad para almacenar este nuevo elemento.
* Un **Underflow** se produce cuando se elimina un elemento de un nodo y este no cumple la cantidad mínima de elementos que debe tener.
* La **Redistribucion** consiste en que un nodo hermano o adyacente le ceda elementos al nodo que está en Underflow.
* La **Fusión** o **Concatenación** consiste en que un nodo se combine con un nodo hermano o adyacente cuando el nodo está en Underflow y no es posible redistribuir.